Fluent Interfaces

Shushobhickae Singh, Chitrangada Nemani
NMIMS-MPSTME, Mumbai, India
Email: {shushobhickae, chitrangada.nmims}@gmail.com

***Abstract*— Modern Enterprise Software Systems entail many challenges such as rapidly changing business scenario, increase in complexity, and shorter time to market and providing business agility without compromising on the quality. Ensuring productivity, quality, consistency, cost effective, reduced cycle time have become a mandate for the teams dealing with modern enterprise software systems. Fluent Interfaces is a powerful technique, which help in taming the programming complexities, reducing boilerplate code, increasing the quality and thereby improving the productivity and cycle time. In this paper we will describe some of our explorations in Fluent Interfaces and why we feel the notion of Fluent Interfaces is a useful technique for enterprise software system. We are currently focusing on two things – a technique for determining fluency of an API and secondly a methodology for designing a fluent interface. We will also share some of the benefits and limitations that we observed during our experimentation. We conclude this paper with a note on the current work that we are doing and the future directions.**

***Key Words* – Fluent Interfaces, Enterprise Software Systems, Application Programmable Interface (API). Domain Specific Languages.**

I. Modern Enterprise software systems and its challenges

The Modern Enterprise Software Systems are growing beyond the boundaries of silo-ed enterprise applications to meet the requirements of rapidly changing markets. They are getting integrated within and across the enterprises thereby metamorphosing into huge ecosystems. This has resulted into immense complexity, which seems to be ever increasing.The complexity of Enterprise Software Systems can be split in a two-dimensional way. The first aspect is the domain in which the software is operating in. The other is the technology, using which we build the solution. Domain has to be treated as an essential complexity, which we cannot avoid. On the other hand, technical complexity is something which we can look into. Today the need of the hour is to bring the technical complexity down there by improving the – productivity, quality, reduce complexity, lesser learning curve, and reduced cycle time, which are the mandates for any enterprise software system. By saving efforts on the technology side we can invest them into domain, there by addressing the business needs in a better and timely manner. Having a good camaraderie between domain and technology, helps us to communicate better with different stake holders which reduces the communication gap and helps in meeting the business demands. Application Programmable Interfaces (APIs) seem to be common IT person's technique for interacting with different components of a software system. The usual APIs have lot of bloated code, need refactoring and re-designing for using them. Consider the case of Java which has been there, for almost 15 years. Enterprises have been extensively using it for almost a decade. This has resulted in the creation of whole bunch of APIs, that instead of leveraging existing APIs, we tend to create new ones to avoid the pain of understanding them. This increase in complexity, adds to the difficulties of developing software as well as testing, maintaining and enhancing them. This overall brings down the PQC factor viz. Productivity, Quality and Consistency as well as increases the complexity, cost and time to market / value. It therefore becomes difficult for enterprise software systems to sustain, evolve or innovate. Fluent Interfaces are a lighter version of Domain Specific Languages which help in overcoming these problems. They help in re-designing APIs to elevate the level of abstraction and making them succinct and easy to use.

II. Fluent Interfaces

Fluent Interfaces (first coined by Martin Fowler and Eric Evans) help in transforming an object oriented API to be more readable, concise, easy to understand and easy to use. They help in developing software which hides the technical complexities and is aligned to the domain. This makes it feasible for non-programmers to understand the context. There has been a wide gap between problem domain and the solution domain as most of the time it is difficult to make out the problem due to lengthy and complicated software. Thus this high level of abstraction helps in increase the productivity, maintainability by bridging the gap between business logic and its development. The cost of adding fluency demands more effort, both in terms of developing and refining it. Making a fluent API involves lot of thought process but the outcome of this investment is worthwhile. Fluent Interface is also often called an internal DSL, since the syntax resembles that of a DSL and it is implemented inside the host language instead of being processed by a parser. Fluent Interface allows us to develop software which is:

- More readable and clean.
- Can be written easily and in shorter period of time.
- Is more concise and succinct for usage
- Easily understandable due to meaningful names
- Avoids redundancies in code, thereby making the code crisper.

Thus, fluent interfaces provide mechanism to develop software where we can readily see the idiomatic patterns, which we may have earlier found it difficult to identify. They also force us to change our perspective on code - it should not be merely functional but also readable, especially if non developers need to consume any aspect of it. Fluent interfaces help in removing unnecessary noise from the software.

DOI: 01.IJIT.01.02.186

III. Examples of Fluent Interfaces

In this section, with the help of an example we will see in detail the effectiveness of using Fluent Interfaces. Apart from this, we have also included a list of the prominent examples which are utilizing Fluent Interfaces.

A. *Using Op4j operation expression*

Consider a scenario where we have to convert a list into an ordered set. The list comprises of Strings which are dates in the DD/MM/YYYY format. Following are the conditions that need to be added:

- Convert them into java.util.calendar objects.
- As some of the strings can be null, so we need to remove them before executing the conversion.
- Also we do not want to include dates in future.

Using "normal" Java code, we can implement the scenario in the following manner:

```
Calendar now= Calendar.getInstance();                    (1)

SimpleDateFormat dateFormat =new SimpleDateFor-
mat("dd/MM/yyyy");                                       (2)
Set<Calendar> set = new LinkedHashSet<Calendar>();(3)
 for (String element : list) {
if (element != null) {

try {
   date = dateFormat1.parse(element);                    (4)
}
catch (ParseException e) {
        throw new SomeException(e);
}

Calendar calendar = Calendar.getInstance();
     calendar.setTimeInMillis(date.getTime());           (5)

        if (!calendar.after(now)) {                      (6)
           set.add(calendar);
        }
     }
}
```

We first get the current calendar date in line (1). The data format is set to DD/MM/YYYY format in line (2). The list is converted to an ordered set in line (3). We then iterate through the list of Strings. If the element i.e. the String is null we do not do any operation and go for the next element. If the String is not empty then we go ahead and parse the element as shown in line (4). We then get the date in a calendar format as shown in line (5). We compare this calendar date with the current date. If it is not beyond the current date then we add it to the set as shown in line (6). Overall, this code is less readable, lengthy and takes more time for execution. This is just an auxiliary code for converting and filtering a list of String dates and it has turned into a lot of complex and hard-to-understand code, probably bigger and uglier than the application logic that might surround it. We use Op4j for the same scenario. It is a Java library aimed at improving quality, semantics, cleanness and readability of Java code, especially auxiliary code like data conversion, structure iteration, filtering, mapping, etc. It allows us to create chained expressions which apply both predefined and user-defined functions to the objects in a fluid and readable way. This improves the way the code looks and greatly reduces the complexity of executing auxiliary low-level tasks in the highly bureaucratic, statically -and strongly- typed language that Java is.

```
Calendar now= Calendar.getInstance();

Set<Calendar> set =
Op.on(list).toSet().map(FnString.toCalendar("dd/MM/yyy
y")).removeAllNullOrTrue(FnCalendar.after(now)).get();
```

In the above code, "Op.on(list).toSet()", converts the list into an ordered set. "map" operation helps the tasks to be executed in an iterative manner. "removeAllNullOrTrue", removes the Strings which are null before they get processed. It also removes cases where the date is greated than the current calendar date by using the function "FnCalendar.after(now)". Thus the above code is crisp, precise and clean as well as readable achieving the same functionality as the "normal" java code which was written earlier. The number of lines of codes look immensely reduced and thereby it becomes easy to manage. Like Op4J there are many more libraries which have Fluent Interface implementation.

B. *Some more examples*

Following table shows some examples in the field of Fluent Interfaces.

Areas in the Enterprise	Non-Fluent Interface	Fluent Interface
Collections	Java Collections from JDK (java.util.Collection)	Lamdaj [7] Op4J [8]
Date and Time	Java Date and Time from JDK (java.util.Date, java.sql.Date, java.sql.Time)	Joda-Time [9]
Performing auxiliary tasks like - Regular Expression, data conversion, string handling, calendar operation, math operations	Java String, Math, Date, Time etc from JDK Java.lang.Math; java.util.Collection; java.util.Comparator;	Op4J [8]
Unit Testing	Java mock framework	JMock [10]
Object relation mapping for Microsoft dotNet	NHibernate	Fluent NHibernate [11]
Managing relational data in application	Java Query Language	Java Persistent API [12]

DOI: 01.IJIT.01.02.186

More and more frameworks, toolkits and libraries are using Fluent Interfaces. The rush for fluent interfaces is not limited to Java, but is also getting widely spread across other technologies like C#, dotNet, php, JavaScript etc. Based on the examples provided above, the sprawl of Fluent Interfaces seems to be spreading across different areas of Enterprises as well as different technologies. Currently we are engaged in identifying some more examples of fluent interfaces to understand their impact on enterprise software systems.

IV. Methodology for Determining extent of Fluency and Designing Fluent Interfaces

Fluent interfaces are a powerful technique to tame the complexity of modern software systems. It can be applied to APIs which are being heavily used. Fluency can be accomplished only after several polishing efforts of the API in concern. Identifying the degree of fluency or designing a fluent interface is not a standard mechanical process but it needs to evolve with usage. Currently, we are exploring some more implementations of fluent interfaces to identify a generic approach for determining the degree of fluency in an API. Based on the explorations that we have done so far, we have come up with a list of characteristics which will help us to determine whether an API is fluent enough or not:

- The API needs to be verbose enough to depict the functionality that is assigned.
- The software has to be clear enough so that even non-programmers can understand the logic.
- There should be no repeated patterns occurring across the software else they will have to be segregated out.
- The software should have sufficient modularity so that there is proper separation of concerns.
- There should be no language restrictions or barriers for using techniques like method chaining etc.
- The software should be succinct enough and there should be no further ways of reducing the lines of code.

If all these characteristics are taken care then the degree of fluency should be high. In case if the APIs are not fluent they can be made fluent using the following mechanisms:

- Have thorough understanding of the API which needs to be made fluent. The API should have been heavily used to understand the problem areas.
- Identify the gaps or areas in an API which can be good candidates for applying fluency.
- Select a method or combination of methods for making the API fluent - Method Chaining, Named Parameters and Factory Classes.
- Provide meaningful names to functions and parameters so that they are in sync with the domain they are associated with.
- In order to make the code more readable and comprehendible, define new functions with names like on, having, of, for, etc.
- Decide the order in which each function will be called and the return parameters, so that it becomes a meaningful transaction.

Following are the standard methods for adding fluency to an API are:

1. Method Chaining – This is the most popular way of implementing fluent interface. A function always returns some object that allows for further calls to be made. Example:

```
HardDrive hd = new HardDrive()
        .capacity(150)
        .external()
        .speed(7200);
```

Here an object of class HardDrive is created by chaining methods: capacity, external, speed.

2. Factory Classes - Provides methods to manufacture the instances required. It is used in cases where there is a need to build up series of related objects. Example:

```
class Complex {

  public static Complex fromCartesian (double real,
 double imag) { return new Complex(real, imag);}

  public static Complex fromPolar (double modulus,
double   angle) {
     return new Complex(modulus * cos(angle), modu-
lus * sin(angle));
   }

    private Complex(double a, double b) {  //...    }
}
Complex c = Complex.fromPolar(1, pi);
```

Here, “fromCartesian(a,b)” and “fromPolar(a,b)” are factory methods of factory class “Complex”. Factory methods are used for disambiguation

```
Generally functions are called as:
myDoc.Save (“SampleFile.txt”,  FilePermissions.All);

Using Named parameters:
myDoc.Save(toFile: "SampleFile.txt", withPermissions:
FilePermissions.All);
```

3. Named Parameters - It clearly states the name of each parameter within the function call itself. It provides a way to include additional “syntax” in a method call thereby improving the readability. Example:

V. Benefits and Limitations

Fluent Interfaces, aim at bringing down the complexity in software making it more readable, manageable and economical thus meeting the targets of PQC. It needs to be noted that Fluent Interfaces are not a silver bullet or panicky enough that it will work in all scenarios. It has been tried and tested

DOI: 01.IJIT.01.02.186

and works well in many cases. Thus we should make a conscious decision while going for designing fluent interfaces. Some of the key benefits of using Fluent Interfaces are:

- It is more readable and easy to understand. It elevates the level of abstraction and thereby hides the complexities from the user, thus making it succinct to use.
- It ensures concise code and avoids redundancies. This helps in saving efforts, which improves the productivity figures and helps in achieving time to market.
- Focuses on making the language ubiquitous from non-programmer's point of view. This helps in avoiding too many feedback sessions with different stakeholders thus narrowing the communication gaps.
- Ease of usage and understanding helps in bringing down the efforts involved in development, testing, maintenance and enhancement which in all lowers the overall cost of the enterprise software system.

Though the benefits of Fluent Interfaces are high but making an API fluent is a difficult and time taking exercise. Following are some of the limitations of developing Fluent Interfaces.

- Identifying the exact problem areas and gaps in an API.
- In order to make an API Fluent, we need to have lots of iterations to ensure the solution is proper and effective.
- Having lots of iterations needs additional efforts and time, which increases the overall cost.
- Coming up with domain flavor for naming convention requires good understanding of the domain itself.

VI. Further Work

Given the observation that Fluent Interfaces are spreading rapidly in enterprise software systems, we are planning to explore some more Fluent APIs. As a part of our early experiments, we have designed fluent interfaces using Dasein Cloud API. The sprawl of Fluent Interfaces can be seen from their usage in areas as common as collections to emerging technologies like cloud APIs. Implementing Fluent Interfaces demands heavy usage of API and lot of iterations for finding the right fluency. Thus it requires a systematic approach for determining whether a given API is fluent enough or not and also for designing Fluent Interfaces. By method of induction, we intent to develop a generic process for determining the fluency of an API and design a methodology for implementing Fluent Interface. Though Fluent Interfaces may be an unstated deliverable but the representative numbers that can be achieved in improving the productivity mandates us to explore this technique further. Fluency can be applied to some of the latest and emerging technologies like Cloud API, Mobile API, Collective intelligence API, REST Interfaces etc. which are becoming very popular. Though our current scope of work is limited to Fluent APIs, but the notion of fluency is creating waves in different programming languages too e.g. FluentJava. The power of fluency seems to be engulfing enterprise software systems and it may not be long when fluent interfaces may become an inherent standard design technique.

Acknowledgment

The work reported here was carried out under the aegis of Innovation Research Lab (IRL) at Patni Computer Systems, under the guidance of Sunil Joglekar, Senior Technical Architect and A.Nagalakshmi, Manager (Software).

References

[1] DSL by Martin Fowler (http://www.martinfowler.com/bliki/DomainSpecificLanguage.html).
[2]DSL wiki (http://en.wikipedia.org/wiki/Domain-specific_language)
[3]Method Chaining (http://martinfowler.com/dslwip/MethodChaining.html).
[4]Method Chaining (http://stackoverflow.com/questions/293353/fluent-interfaces-method-chaining)
[5]Factory Pattern (http://en.wikipedia.org/wiki/Factory_method_pattern)
[6]Named Parameters (http://en.wikipedia.org/wiki/Named_parameter).
[7]Lamdaj (http://code.google.com/p/lambdaj/)
[8]Op4j (http://www.op4j.org/basics.html)
[9]Joda-Time (http://joda-time.sourceforge.net/)
[10]JMock (http://www.jmock.org/)
[11]Fluent nHibernate (http://wiki.fluentnhibernate.org/Getting_started)
[12]Java Persistent API (http://en.wikipedia.org/wiki/Java_Persistence_API)

About the Authors

Shushobhickae Singh is a student of B.E (Computer Science) final year from NMIMS-MPSTME, Mumbai. She has carried out the work on Fluent Interfaces in Patni Computer Systems, as a part of the partial fulfillment of Bachelor of Engineering degree.

Chitrangada Nemani is a student of B.E (Information Technology) final year from NMIMS-MPSTME, Mumbai. She has carried out the work on Fluent Interfaces in Patni Computer Systems, as a part of the partial fulfillment of Bachelor of Engineering degree.

DOI: 01.IJIT.01.02.186